

## Auxiliar 3 - Análisis matemático de algoritmos

Profesores: Nelson Baloian

Patricio Poblete

Auxiliares: Manuel Cáceres

Sebastián Ferrada

Sergio Peñafiel

### P1. StoogeSort.

StoogeSort es un algoritmo de ordenación de arreglos que se basa en el principio de dividir para reinar. Su funcionamiento para ordenar un arreglo  $A$  de tamaño  $n$  es el siguiente:

- Se analiza el primer y el último elemento del arreglo, si el primero es menor se intercambia con el último.
- Se divide el arreglo  $A$  en 3 sub-arreglos de tamaño  $n/3$
- Se aplica StoogeSort sobre los primeros  $2n/3$  elementos del arreglo.
- Se aplica StoogeSort sobre los últimos  $2n/3$  elementos.
- Se vuelve a aplicar StoogeSort sobre los primeros  $2n/3$  elementos del arreglo

A partir de lo anterior:

- (a) Implemente la función `public void StoogeSort(float[] A, int inicio, int fin)` que ordena según StoogeSort
- (b) Analice la complejidad del algoritmo, discuta sobre este resultado comparándolo con BubbleSort y QuickSort.

### P2. Similitud genética (Subsecuencia común más larga)

El ADN puede ser representado como una secuencia de strings (A,C,G,T), las secuencias de distintas bases nitrogenadas(caracteres) definen genes en el ADN. En biología un problema de gran interés es encontrar la similitud entre distintas hebras de ADN. Para esto debemos encontrar cuán parecido son 2 strings  $X$  e  $Y$ , una de medida de similitud consiste en encontrar la subsecuencia común más larga a ambas.

Se define una subsecuencia de la siguiente forma: sean  $X$  e  $Y$  2 strings cualquiera diremos que  $X$  es subsecuencia de  $Y$  si podemos obtener  $X$  al extraer 0 o más caracteres de  $Y$ . (Observe que no es lo mismo que ser substring). Por ejemplo, "electra", "once", "grita" son subsecuencias de "electroencefalografista".

Para encontrar esta subsecuencia más larga podemos revisar todas las subsecuencias de  $X$  y comprobar si también es subsecuencia de  $Y$  retornando la más larga que cumpla esta propiedad.

Sin embargo, esta solución revisa cada una de las  $2^{|X|}$  subsecuencias de  $X$ , por lo que el algoritmo es ineficiente.

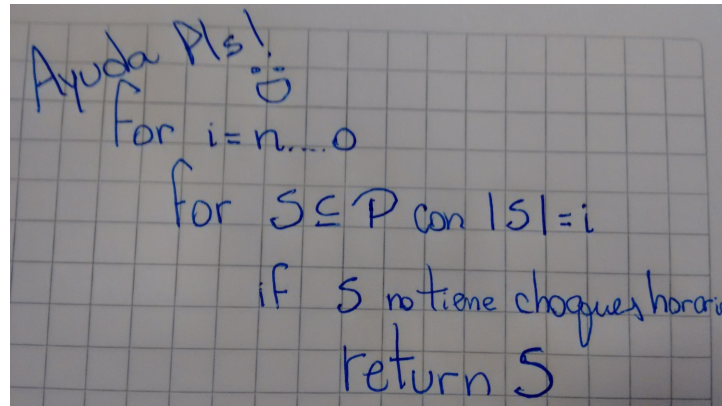
Proponga soluciones más eficientes a esta, e implemente la mejor.

**Hint:** La solución del problema posee subestructuras óptimas (Programación Dinámica)

### P3. Maratón de películas! (Maximización de tareas)

Alicia y Roberto quieren ver la mayor cantidad de películas este fin de semana. Para esto Roberto revisa la programación de *TODOS* los canales de televisión obteniendo un conjunto de  $n$  películas  $\mathcal{P} = \{[i_0, f_0], \dots, [i_{n-1}, f_{n-1}]\}$  (donde  $i_k$  y  $f_k$  corresponden a los tiempos de inicio y fin de la película  $k$ ) y le pide a Alicia que escoja la mayor cantidad de películas de modo que puedan verlas todas de inicio a fin.

Alicia no sabe programar pero considera que el siguiente algoritmo(escrito en pseudocódigo) funcionaría:



```
Ayuda Pls! ☹️
for i=n...0
  for S ⊆ P con |S|=i
    if S no tiene choques horario
      return S
```

Para lo que sigue considere que los tiempos de  $P$  están en una matriz de  $n \times 2$  y solo retorne la cantidad de películas escogidas.

- Implemente el pseudocódigo escrito por Alicia como una función  
`public int maximoPelículas(int [][] P)`
- Considere que es Viernes (poco tiempo para el fin de semana) y que Roberto recolectó muchas películas ¿Qué problema tiene esta solución?
- Implemente nuevamente la función, pero más eficientemente.  
**Hint:** Sea avaro
- Propuesto:** Modifique las funciones anteriores de modo que retornen un `int []` que tenga los índices en  $P$  de las películas escogidas.